

基于 MemoryNetwork 的推特流行度预测

盛俊杰，徐洪义，黄金根

10152510150, etc

1 问题描述

这次大作业我们小组选择的是推特流行度预测问题，通过用户之前发出的推特的内容，标签，转载数，以及用户自身的 follower_cnt 和 statuses_cnt 来预测一条新发的推特的最终转载数。

2 数据分析

我们所获得数据如下

- tweet_train.json / tweet_test_unlabel.json[list]
 - id[int]: tweet ID
 - user_id[int]: author ID
 - text[str]: raw text description
 - hashtags[list] : hashtags
 - retweet_count[int] (**train only**): retweet count within three days
 - score[float] (**train only**): popularity score, $\log_2(\text{retweet_count})$
- tweet_user.json[dict]
 - key[int]: author ID
 - value[dict]:
 - followers_count[int]: number of followers
 - statuses_count[int]: number of statuses

cc

需要预测的任务是

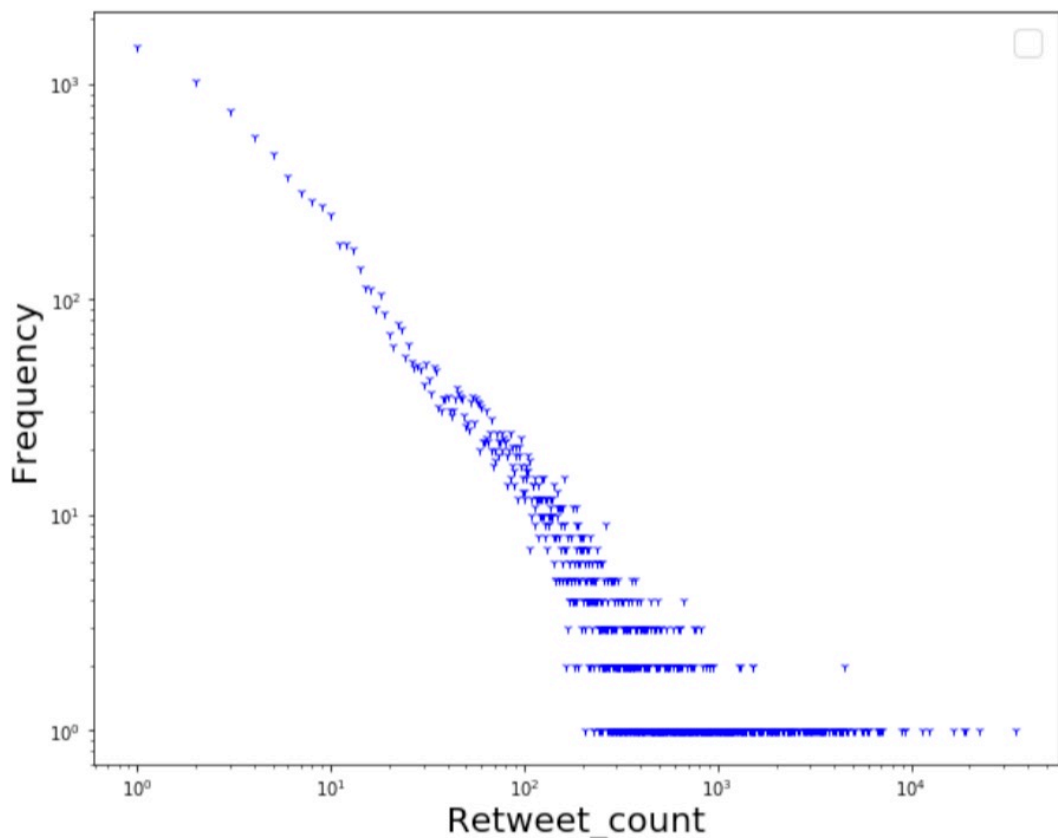
- tweet_test_labeled.json
 - key[int]: tweet ID
 - value[float]: popularity score

cc

可以看到我们需要预测的是一条 tweet 的 *popularityscore*，而可以用来建立模型的特性有 *userid*, *text*, *hash_tags* 以及用户的特征 *followers_count*, *statuses_count*。大致先看下各个属性的特征好了：

1. *text*: 文本数据，包含特殊字符 (需要去除)，长度一般在 60words 以内
2. *userid*: 用户 id，训练集中有总共 134 个用户
3. *hash_tags*: 标签，有 413 个标签
4. *followers_count*: 关注者数目 50W ~ 3500W 之间
5. *statuses_count*: 发布状态数 1W ~ 165W 之间

再来看一下 *retweet_count* 的分布：



cc

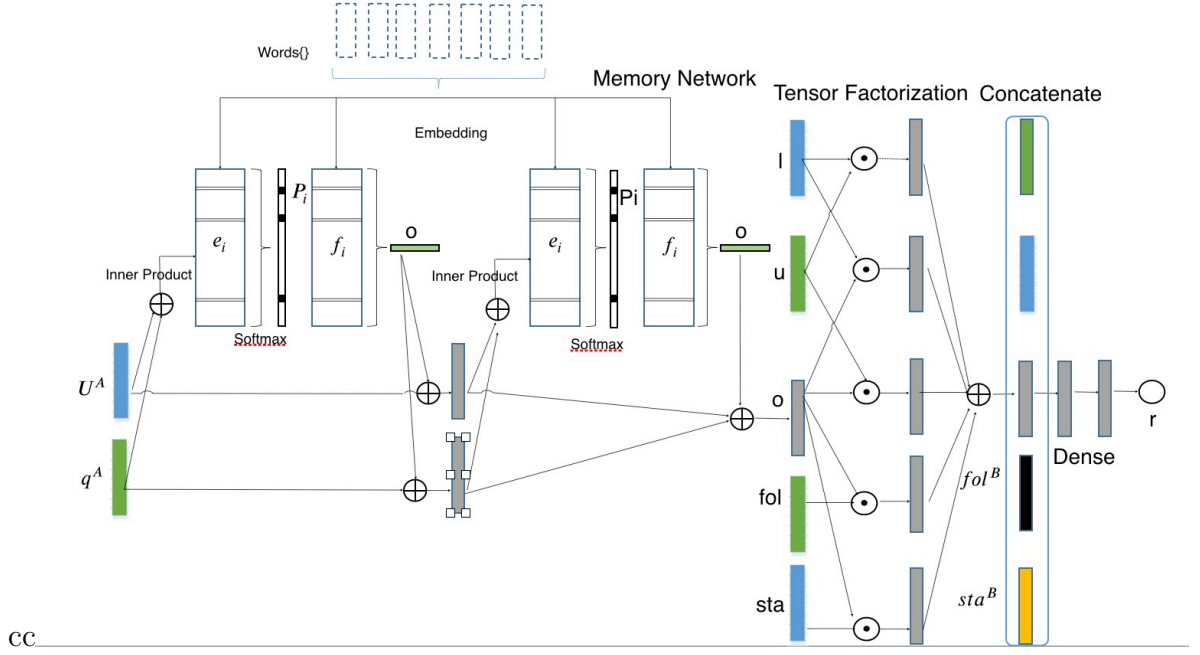
可以看出高转发数的 tweet 出现频度是很低的，而大部分的转发数都是维持在较低的水平。

3 模型阐明

在这里我们采用的方法是使用 Memory NetWork 与张量分解结合形成的一个端到端的模型，原文中称之为 MOOD 模型。使用 Memory NetWork 获得文本的最佳表示，

3.1 模型总览

整个模型的输入由 $userid$, $text$, $hash_tags$, $followers_count$, $statuses_count$ 组成，核心可分为 Memory Network 与 Tensor Factorization 两部分。模型图见下



3.2 Memory Network

3.2.1 Memory Representation

在这个模块中，MOOD 分别定义了 memories for word,user 和 hash_tag。我们设字典大小为 $|V|$ ，文档长度为 L ，嵌入维度为 k （这里又进一步假设 $userid$, $hash_tag$ 的嵌入维度也为 k ），我们将每一个单词都与 user 和 $hash_tag$ 相关联，同时又为了考虑词序信息，我们将两个绝对位置编码矩阵 $E^p \in \mathbb{R}^{k \times L}$ 和 $F^p \in \mathbb{R}^{k \times L}$ 合并到基本词嵌入中。同时，MOOD 定义 user 和 $hash_tag$ 的注意嵌入矩阵 $U^A \in \mathbb{R}^{k \times |Q|}$ 。那么有 $u_u^A = U_{:,u}^A$ 和 $q_q^A = Q_{:,u}^A$

3.2.2 Attention for Memory

在原始词汇空间中，文本介绍可以表示为一个热编码的序列（虽然在实现时我们是直接记录词典索引序列的，但为了论文描述方便）。对于引言 d 中的第 j 个单词 w_j ，热编码向量表

示为 $\hat{w}_j \in \{0, 1\}^{|V|}$ 。假设 $v(j)$ 示词汇表中 w_j 的索引, 我们可以得到嵌入 $e_{v(j)} = E\hat{w}_j + E_{:,j}^p$ 以类似的方式, 也可以获得 $f_{v(j)}$

基于这些嵌入, MOOD 通过以下等式计算关于 $v(j)$ 的 user u 和 hash_tag q 的关注权重:

$$w_{v(j)}^{u,q} = (u_u^A + q_q^A)^T e_{v(j)}$$

这样我们就同时考虑进去了 user 和 hash_tag 两个特征。

3.2.3 Output Representation of Text

内存网络模块的核心目标是获得更好的活动介绍表示。首先计算 $p_{v(j)}^{u,q} = \text{softmax}(w_{v(j)}^{u,q})$ 代表 $v(j)$ 表示 d 的重要性的概率。单层记忆网络中学习到的 d 的表示记为 o :

$$o = \sum_j p_{v(j)}^{u,q} f_{v(j)}$$

3.2.4 Multi-layered Extension

类似于深度记忆网络采用的共同策略, MOOD 更新了每层之间的组织者和位置嵌入。Organizer u 和 location q 在第 k 层的嵌入分别表示为 $u_u^{A,K}$ 和 $q_q^{A,k}$ 。对于第一层, 有 $u_u^{A,1} = u_u^A$ 和 $q_q^{A,1} = q_q^A$ 。按如下公式化迭代更新:

$$u_u^{A,K+1} = u_u^{A,K} + o^k$$

$$q_q^{A,K+1} = q_q^{A,K} + o^k$$

3.3 Tensor Factorization

在张量分解模块中, MOOD 定义了 *userid, text, hash_tags, followers_count, statuses_count* 的交互嵌入。它将五种嵌入模型组合在一起, 以捕捉它们对活动流行度的共同影响。这个交互嵌入其实可以有怪多种定义方式, 这里使用简单的乘积求和表达。公式如下:

$$\psi_d^{u,q} = u_u^I \odot q_q^I + u_u^I \odot o_d + q_q^I \odot o_d + f_f^I \odot o_o^I + s_s^I \odot o_o^I$$

通过经验可知 (玄学), *userid, hash_tags*, 对于评分还是有很明显的偏置作用的。于是引入用户 u 和标签 q 的偏置嵌入 u_u^B 和 q_q^B 我们进一步连接偏置嵌入和集成嵌入, 并将它们与完全连接层相关联以计算流行度 \hat{r} :

$$\hat{r} = \Theta^T \sigma(W_1^T [\psi_d^{u,q}; u_u^B; q_q^B] + b_1) + b$$

这里我们对其进行了改进，考虑到部分 user 在训练集出现次数较低，我们可能无法通过那少量的数据习得用户的特征，所以在 concat 时我们加入 *followers_count*, *statuses_count* 两层，来更好的更话用户特征对流行度的预测。而在全连接层上我们选择了 2 层的隐层（全连接），再全连接到节点数为 1 的输出层。

4 代码构建

4.1 预处理

首先我们对 text 进行处理，首先进行预处理，去除链接与特殊字符，再进行词典编码，为了保留词序信息，这里我们按顺序记录每个词语的 index。实现如下

```
1 # judge if containAlpha
2 def containAlpha(word):
3     if len(word) == 0:
4         return False
5     else:
6         if re.search('[a-z]', word):
7             return True
8         else:
9             return False
10 # delete uri info
11 def deleteUri(text):
12     textlist = text.split()
13     resultlist = []
14     for word in textlist:
15         if word.startswith('http'):
16             continue
17         if word.startswith('#'):
18             resultlist.append(str(word[1:]))
19         if containAlpha(word):
20             resultlist.append(word)
21     return ' '.join(resultlist)
22
23 def textPrecessing(text):
24     text = text.lower()
25     text = deleteUri(text)
26     return text
```

```

27
28 from sklearn.feature_extraction.text import CountVectorizer
29 vectorizer =CountVectorizer(stop_words='english')
30 for item in data:
31     t=textPrecessing(item['text'])
32     #corpus is new text index matrix
33     corpus.append(t)

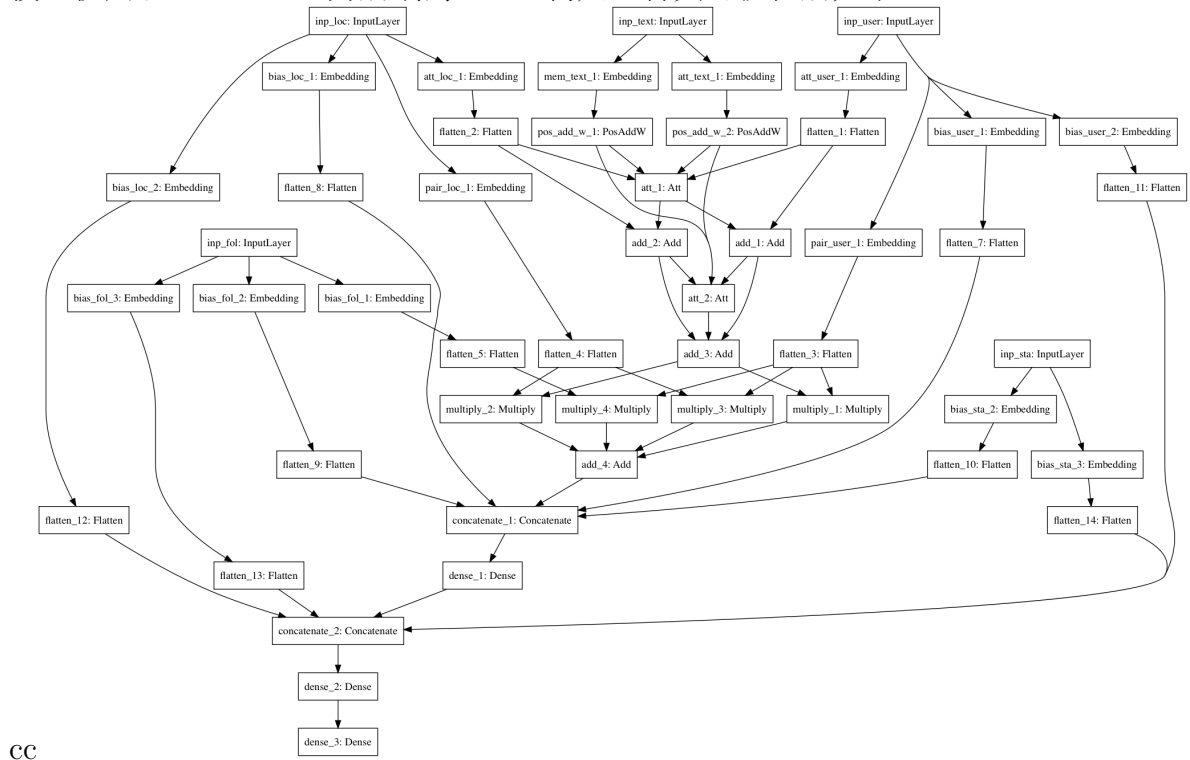
```

对于 hashtags 的处理同上

而对于 *followers_count*, *statuses_count*, 我们将其分段来表示。

4.2 模型构建

模型使用以 tensorflow 为后端的 keras 构建，将其可视化后如下



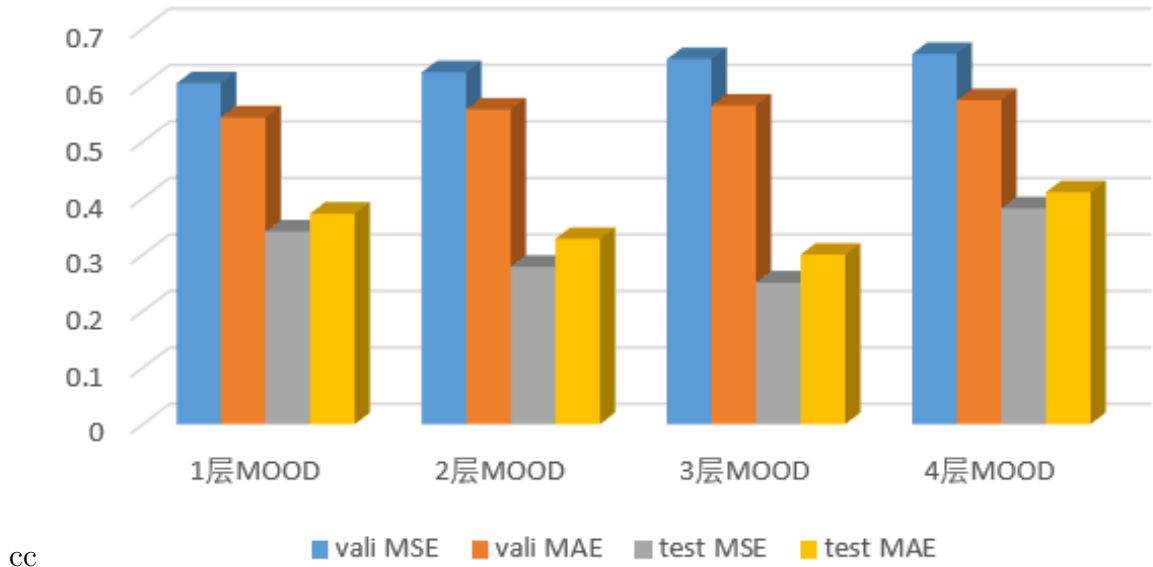
5 实验过程

因为网络中的层数，隐层节点数，嵌入维度等都是一个个超参数，而这些超参对模型性能有一定的影响，所以在这里我们简要的记录我们的实验过程。

cc

## 实验过程							
	vali		test			# 参数说明	
	MSE	MAE	MSE	MAE		## 数据集划分	
1层MOOD	0.6035	0.5432	0.3409	0.3727		all:7952	
2层MOOD	0.6228	0.557	0.2787	0.3282		train:6090	
3层MOOD	0.6462	0.5637	0.2508	0.3		vali:1862	
4层MOOD	0.6555	0.5735	0.3821	0.4103		test:7952	

实验过程



Reference